J. Symbolic Computation (1990) **9**, 423–427

Automated Proof of Ring Commutativity Problems by Algebraic Methods

HANTAO ZHANG
Department of Computer Science
The University of Iowa
Iowa City, IA 52242
hzhang@herky.cs.uiowa.edu

(Received July 4, 1989)

Using a simple computer program, we have proved thousands and thousands of instances of the theorem that for any integer $n > 1$, for any element x in an associative ring, $x^n = x$ implies the commutativity of the ring (an instance of the theorem is obtained by taking a specific value for n). The program is based on Newton's binomial theorem and Euclid's *gcd* algorithm. New algorithms are introduced to speed up the *gcd* computation and to decide quickly whether a binomial coefficient is odd.

1 Introduction

N. Jacobson (1945) solved the open problem that an associative ring is commutative if for every element x in the ring, there is an $n > 1$ such that $x^n = x$. A weak version of this theorem is that for any integer $n > 1$, for every element x in an associative ring, if $x^n = x$, then the ring is commutative. By taking $n = 2$, 3, and so on, in the weak version, we obtain a family of problems for testing automated theorem proving programs. We will refer each member in this family as *the ring problem of case i*, or simply the case *i* problem.

The case 2 problem is easy to prove. However, the case 3 problem has been considered as one of the most challenging problems for automated reasoning programs (see for instance, Bledsoe (1977), Lusk & Overbeek (1985), Wos (1988)). To the best of our knowledge, only four computer proofs of the case 3 have been reported (Veroff (1981), Stickel (1984), Wang (1987), Kapur & Zhang (1989)). Wang (1987) also reported a proof of the case 4. No computer proofs are known for the case 5. In Kapur & Zhang (1989), we reported a proof of the case 6. These facts illustrate the great difficulty of attacking this family of problems by computer programs.

By studying the proof of the case 6 problem, we realized that a useful technique can be applied to the whole family of the ring problems. This technique is just based on some methods known from high school algebra, such as computing binomial equation coefficients and the greatest common divider (*gcd*) of two one-variable polynomials. This study then leads to writing an efficient program to prove thousands and thousands of ring problems of this family. By the time this paper was written, the biggest problem we have succeeded in proving is the case $n = 2^{50000}+2$. In this note, we present briefly the technique and some results; a more detailed presentation is given in Zhang (1989).

0747–7171/90/040423+05 $03.00/0

At first, we point out that $x^n = x$ implies $-x = x$ and $2x = 0$ in a ring when n is even. Because these two identities are very useful in proving even number case ring problems, let us consider these cases at first.

Let

$$B(x,y,n) = \sum_{k=1}^{n-1} \delta_{n,k} x^k y^{n-k}, \quad \text{where } \delta_{n,k} = \begin{cases} 1 & \text{if } \binom{n}{k} \text{ is odd} \\ 0 & \text{otherwise} \end{cases}$$

Note that $B(x,y,n)$ can be easily computed (modulo 2, and discard the first and the last item) from Newton's binomial theorem. Note also that Newton's binomial theorem is true only for commutative elements. However, the following can be easily established (Zhang (1989)):

Lemma 1 *If n is even and $xy = yx$, then $B(x,y,n) = 0$ is true in the case n problem.*

Based on the above lemma, our technique can be described as follows: (1) Replacing y by x^2 in $B(x,y,n)$, we obtain an identity $B(x,x^2,n) = 0$, which is true in the case n problem when n is even. (2) Computing $g = gcd(B(x,x^2,n), x^n + x)$. From $B(x,x^2,n) = 0$ and $x^n + x = 0$, we know $g = 0$ is true in the case n problem. (3) If g is equal to either $x + x^2$ or $x + x^4$, we then obtain a computer proof of the case n problem because the computer proofs of the cases 2,4 are known. This idea can be expressed by a simple algorithm:

Algorithm REDUCE(n)

If n is even and $gcd(x^n + x, B(x,x^2,n))$ is either $x^2 + x$ or $x^4 + x$
then print *a computer proof of the case n problem is found*
else print *the method fails.*

The idea is better illustrated by the case 6 problem: $B(x,y,6) = x^4y^2 + x^2y^4$ and $B(x,x^2,6) = x^8 + x^{10}$. The *gcd* of $x^8 + x^{10}$ and $x^6 + x$ is $x^2 + x$ (mod 2). Thus, $x^2 = x$ is true in the case 6 problem. In other words, we "reduce" the case 6 problem to the case 2 problem by the binomial theorem and the *gcd* computation.

n	gcd	n	gcd	n	gcd	n	gcd	n	gcd
2	(2 1)	22	g_1	42	(2 1)	62	(2 1)	82	(4 1)
4	(4 1)	24	(2 1)	44	(2 1)	64	(64 1)	84	(2 1)
6	(2 1)	26	(2 1)	46	(16 1)	66	(2 1)	86	(2 1)*
8	(8 1)	28	(4 1)	48	(2 1)	68	(2 1)	88	(4 1)
10	(4 1)	30	(2 1)	50	(8 1)	70	(4 1)	90	(2 1)
12	(2 1)	32	(32 1)	52	(4 1)	72	(2 1)	92	(8 1)
14	(2 1)	34	(4 1)	54	(2 1)	74	(2 1)*	94	g_2
16	(16 1)	36	(8 1)	56	(2 1)	76	(16 1)	96	(2 1)
18	(2 1)	38	(2 1)	58	(4 1)	78	(8 1)	98	(2 1)
20	(2 1)	40	(4 1)	60	(2 1)	80	(2 1)	100	(4 1)

$g_1 = (10\ 9\ 8\ 3\ 2\ 1)$ $\quad g_2 = (34\ 33\ 32\ 3\ 2\ 1)$

*: it is $gcd(x^n + x, B(x,x^2,n), B(x,x^4,n))$ (see Section 4).

Table 1: First 50 even number case problems

REDUCE has been implemented in Domain/Common lisp on an Apollo 3500. We experimented with it for thousands and thousands of even number n's, ranging from 6 to $2^{50000} + 2$. The polynomial *gcd* computed by REDUCE has more than 55% of chance to be $x^2 + x$ and about

22% of chance to be $x^4 + x$. I.e., REDUCE has 77% of chance to succeed. Table 1 lists the *gcd* computed in REDUCE for the first 50 even numbers between 2 and 100. In the table, (2 1) denotes the polynomial $x^2 + x$, (10 9 8 3 2 1) denotes $x^{10} + x^9 + x^8 + x^3 + x^2 + x$, etc.

3 Implementation Issues

The complexity of REDUCE is dominated by the computation of $B(x, y, n)$ and *gcd*. In the following, we introduce two new algorithms which can speed up REDUCE by a factor of thousands in many cases.

By definition, $B(x, y, n) = \sum_{k=1}^{n-1} \delta_{n,k} x^k y^{n-k}$. Using proper data structure to store $B(x, y, n)$, the only thing needed to compute is $\delta_{n,k}$, for which there exist many methods. For instance, we may use the identity: $\delta_{n,k} = \delta_{n-1,k} + \delta_{n-1,k-1} \pmod 2$. The complexity of this method is $O(n^2)$ in the worst case. We first implemented this method to support REDUCE. When $n = 50,000$, in order to compute $\delta_{n,k}$ for $1 \le k \le n/2$, the program takes more than one week on an Apollo 3500. However, using the algorithm suggested by the following theorem, only 2.5 seconds are needed for the same problem on the same machine.

Theorem 2 *Let* $n = b_m 2^{m-1} + b_{m-1} 2^{m-2} + \ldots + b_2 2^1 + b_1$, *where* b_i *is either* 0 *or* 1 *for* $1 \le i \le m$. *Associated with n, we define recursively a sequence of m binary strings as follows:*

$$w_1 = b_1$$
$$w_{i+1} = \begin{cases} w_i 0^{2^i} & \text{if } b_{i+1} = 0 \\ w_i 1 w_i & \text{otherwise} \end{cases}$$

Then $\binom{n}{k}$ *is odd if and only if the* k^{th} *symbol (from left to right) of* w_{m-1} *is* 1 *for* $0 < k < \lfloor n/2 \rfloor$.

The proof of the theorem can be found in Zhang (1989). The new algorithm consists of computing w_{m-1} according to its definition in the theorem and is of time complexity $O(n)$.

Example 3 To decide the value of $\delta_{22,k}$ for $1 \le k < 11$, we first compute the binary representation of 22, i.e, (1 0 1 1 0), then define the binary words w_1, w_2, w_3 and w_4:

$$\begin{array}{lllll} w_1 = 0 & & & & \text{because } b_1 = 0, \\ w_2 = w_1 1 w_1 & = & 010 & & \text{because } b_2 = 1, \\ w_3 = w_2 1 w_2 & = & 0101010 & & \text{because } b_3 = 1, \\ w_4 = w_3 0^{2^3} & = & 010101000000000 & & \text{because } b_4 = 0. \end{array}$$

Now by the theorem, $\delta_{22,1}$ is 0 because the first symbol of w_4 is 0; $\delta_{22,2}$ is 1 because the second symbol of w_4 is 1. It is easy to see from w_4 that $\delta_{22,2} = \delta_{22,4} = \delta_{22,6} = 1$ and $\delta_{22,k} = 0$ for $1 \le k < 11$ and $k \notin \{2, 4, 6\}$.

The algorithm used in REDUCE to compute *gcd* is a modification of Euclid's algorithm, which is easy to implement, especially when we consider the remainders of polynomials modulo 2. In our implementation, polynomials are represented by a decreasing list of positive integers, that is, the polynomial $x^{a_1} + x^{a_2} + \cdots + x^{a_n}$ is represented by $(a_1\ a_2\ \cdots\ a_n)$. Let $f_i = (a_1\ a_2\ \cdots\ a_n)$ and $f_j = (b_1\ b_2\ \cdots\ b_m)$ and $d = a_1 - b_1 \ge 0$, then a subtraction in Euclid's algorithm can be obtained by merging the two lists $(a_2\ \cdots\ a_n)$ and $((b_2 + d)\ \cdots\ (b_m + d))$ and then removing pairs of identical integers from the merged list.

However, the time complexity of this algorithm may be up to $O(n^2)$ because a subtraction may take $O(n)$ steps and there may be $O(n)$ steps of subtractions in the worst case. The following theorem suggests that some of substructions can be saved.

Theorem 4 *Let $f_1 = x^{i_1} + A$, $f_2 = x^{j_1} - x^{j_2}$ and $i_1 \geq j_1 > j_2$, then $f_3 = -x^{i_1-k*(j_1-j_2)} + A$, satisfies the relation $f_1 = h * f_2 + f_3$ for some h, where $k = \lfloor (i_1 - j_2)/(j_1 - j_2) \rfloor$.*

Proof. Let $h = x^{i_1-j_1} + x^{i_1-2*j_1+j_2} + \ldots + x^{i_1-k*j_1+(k-1)*j_2}$. □

The relation $f_1 = h * f_2 + f_3$ implies that the *gcd* of f_2 and f_3 is that of f_1 and f_2. For example, suppose $f_1 = x^{100} + x$ and $f_2 = x^2 - x$, we have $k = 99$ and $f_3 = -x^{100-99*(2-1)} + x$, which is equal to 0. Hence, $x^2 - x$ is the *gcd* of f_2 and f_3 or that of f_1 and f_2. Note that it needs 100 substructions for Euclid's algorithm to derive this result.

The technique suggested by the above theorem was integrated in REDUCE and improves the performance of REDUCE by up to a factor of thousands in many examples. For instance, REDUCE takes only 1.5 hours to compute the $gcd(x^n + x, B(x, x^2, n)) = x^2 + 1$ for $n = 2^{50000} + 2$ ($B(x, x^2, n) = x^{2^{50001}+2} + x^{2^{50000}+4}$ and 99.9% of the time is spent on the garbage collection).

It will be interesting to see how this technique can be extended to the general case where f_2 has more than two items. It has been observed in Buchberger (1987) that Gröbner basis algorithms subsume Euclid's *gcd* algorithm. Hence, it is also interesting to see how the above technique can be applied in Gröbner basis algorithms.

4 Discussion

We have shown that using the binomial theorem and the commutativity of special elements like x and x^2 in an associative ring, some new identities may be obtained; using the Euclid's *gcd* algorithm, a ring problem can be reduced to a simpler ring problem of which computer proofs are known. By this way, we obtain computer proofs for many ring problems of even number case.

Our technique can apply to odd number cases, too. Let $N(a, b, n) = \sum_{k=1}^{n-1} \binom{n}{k} x^k y^{n-k}$, then the identity $N(x, x^2, n) = 0$ is true in the case n problem, no matter whether n is even or not. For the case 5 problem, during the computing process of the *gcd* of $x^5 - x$ and $N(x, x^2, 5)$, the polynomials $5x^3 - 5x$, $15x^2 - 15x$ and $30x$ can be produced, from which we obtain the following equalities:

$$5x^3 = 5x, \qquad 15x^2 = 15x, \qquad 30x = 0.$$

These equalities can be considered as useful lemmas to prove the case 5 problem. With the aid of these lemmas, we are able to work out a proof of the case 5 by hand. One of difficulties in proving the ring problems of odd number case is that the identities like $2x = 0$ and $-x = x$ do not hold in these cases. Without these identities, the *gcd* computation is very expensive and its result is not of form $x^k + x$ in general. For instance, the *gcd* of $x^7 - x$ and $N(x, x^2, 7)$ is $14x^5 + 14x^4 - 14x^2 - 14x$.

As illustrated in Table 1, our technique cannot solve all the even case problems. For instance, REDUCE cannot prove the case 2^i problems because $B(x, y, 2^i)$ is identical to 0. For the cases when $n = 22$, 94, etc., the degree of *gcd* is lower than the corresponding $x^n + x$, but these polynomials do not correspond to the special hypothesis of any ring problem.

Because the goal of the algorithm REDUCE is to reduce the polynomial $x^n + x$ to either $x^2 + x$ or $x^4 + x$ by computing $g_1 = gcd(x^n + x, B(x, x^2, n))$, if such a goal fails, we may reduce further g_1 by applying the same technique. For instance, we may compute $g_2 = gcd(g_1, B(x^j, x^k, n))$ for some j and k. If g_2 is either $x^2 + x$ or $x^4 + x$, we then succeed. This generalization does help to solve two problems ($n = 74, 86$) among the 50 ring problems listed in Table 1. For instance,

$$\begin{aligned} g_1 &= gcd(x^{74} + x, B(x, x^2, 74)) = x^{20} + x^{18} + x^{12} + x^9 + x^3 + x \\ g_2 &= gcd(g_1, B(x, x^4, 74)) = x^2 + x. \end{aligned}$$

However, this generalization is not effective in practice. For the case 22 problem, $g_1 = gcd(x^{22} + x, B(x, x^2, 22)) = x^{10} + x^9 + x^8 + x^3 + x^2 + x$. We have tried thousands of different pairs $\langle j\ k \rangle$ to

compute $gcd(g_1, B(x^j, x^k, 22))$, the result is always g_1 itself. Because of this, we conjecture that $x^{10}+x^9+x^8+x^3+x^2+x$ is a factor of $B(x^j, x^k, 22) = x^{20k+2j} + x^{18k+4j} + x^{16k+6j} + x^{6k+16j} + x^{4k+18j} + x^{2k+20j}$. However, to the best of our knowledge, there do not exist such algorithms. So, we leave it as an open problem:

Problem 5 *Is there any algorithm to compute the gcd of two polynomials of which the degree itself is a (linear) polynomial over integers?*

The limitation of our technique is mainly due to the fact that our program performs only *substitution of equals for equals* and *instantiation of variables* such simple inference rules. Jacobson's result tells us that each ring problem can be proved by these simple inference rules and our program does find such proof for some problems. In spite of the limitation mentioned above, we feel that the research on this particular problem is worthwhile and profitable. By studying this problem, we were able to design new algorithms and have a better understanding of ring structures; see Zhang (1989) for more details.

Acknowledgement: Many thanks to Deepak Kapur for his persistent support during the course of this work. I am also grateful to Rusty Lusk, Dave Musser, Ross Overbeek, Rick Stevens, Mark Stickel, Tie-Cheng Wang, and Larry Wos for their encouragement and enthusiasm shown for this work.

References

Bledsoe, W., (1977) Non-resolution theorem proving, *Artificial Intelligence* 9, 1, pp.1-35.

Buchberger, B., (1987) History and basic features of the critical-pair/completion procedure. *J. Symbolic Computation* 3, 3-38.

Jacobson, N., (1945) Structure theory for algebraic algebras of bounded degree. *Annals of Mathematics 46*, 4, 695-707.

Kapur, D., and Zhang, H. (1989) A case study of the completion procedure: proving ring commutativity problems *unpublished manuscript*, Dept. of CS, University of Iowa.

Lusk E., and Overbeek, R. (1985). Reasoning about equality. *J. of Automated Reasoning*, 6, 209-228.

Stickel, M. (1984) A case study of theorem proving by the Knuth-Bendix method: Discovering that $x^3 = x$ implies ring commutativity. Proc. of *7th Conf. on Automated Deduction*, NAPA, Calif., LNCS 170, Springer Verlag, 248-258.

Veroff, R.L. (1981) Canonicalization and demodulation. Report ANL-81-6, Argonne National Lab., Argonne, IL.

Wang, T.C. (1988) Case studies of Z-module reasoning: Proving benchmark theorems from ring theory. *J. of Automated Reasoning*, 3.

Wos, L. (1988) *Automated Reasoning: 33 basic research problems.* Prentice Hall, NJ.

Zhang, H., (1989) Prove ring commutativity problems by algebraic methods, *Tech. Report* 89-04, Dept. of Computer Science, University of Iowa.